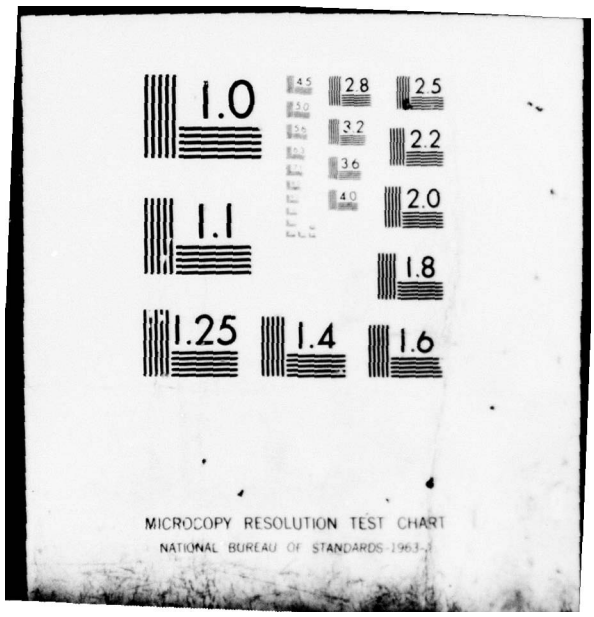
AD-A068 237 CARNEGIE-MELLON UNIV PITTSBURGH PA MANAGEMENT SCIENC--ETC F/G 12/2
A SINGLE SOURCE TRANSPORTATION ALGORITHM.(U)
FEB 79 R V NAGELHOUT, G L THOMPSON N00014-75-C-0621
UNCLASSIFIED MSRR-429 NL

| OF |
AD
A068237

END
DATE
FILMED
6 -79
DDC

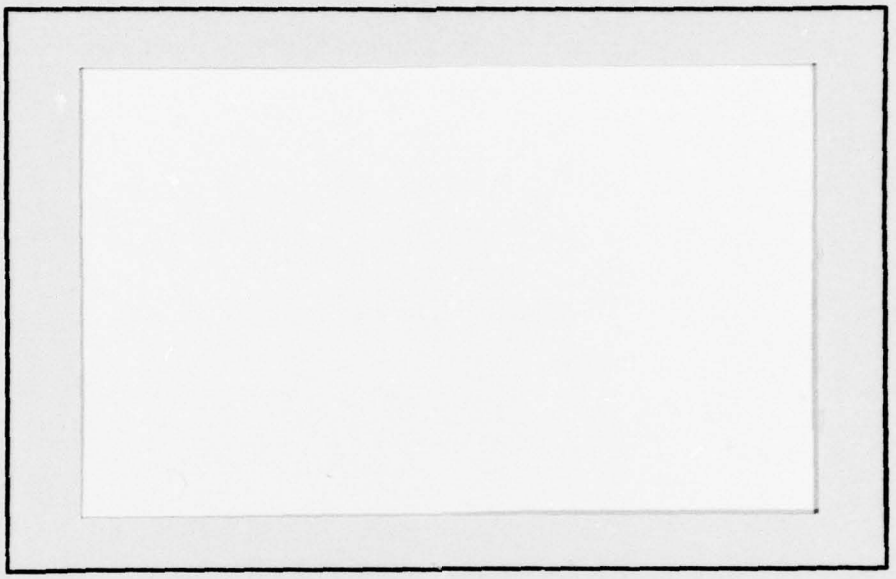

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LEVEL II

(1)

AD A068237

DDC FILE COPY



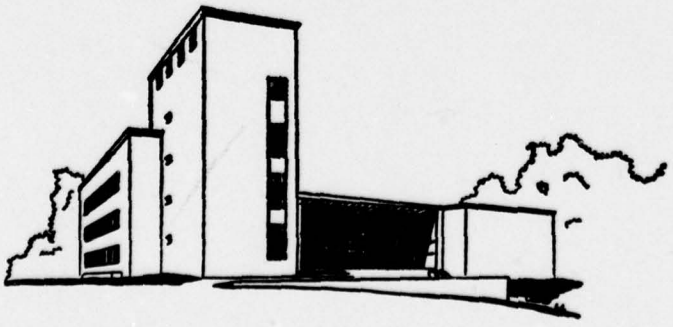
DDC
RECEIVED
MAR 8 1979
C

Carnegie-Mellon University

PITTSBURGH, PENNSYLVANIA 15213

GRADUATE SCHOOL OF INDUSTRIAL ADMINISTRATION

WILLIAM LARIMER MELLON, FOUNDER



This document has been approved for public release and sale; its distribution is unlimited.

79 04 30 100

1473

W.P. #41-78-79

AD A068237

DDC FILE COPY

Management Science Research Report No. 429

A SINGLE SOURCE TRANSPORTATION ALGORITHM

by

Robert V. Nagelhout and Gerald L. Thompson
Carnegie-Mellon University

February 1979

DDC RECEIVED MAY 3 1979 C

This report was prepared as part of the activities of the Management Sciences Research Group, Carnegie-Mellon University, under Contract N00014-75-C-0621 NR 047-048 with the U. S. Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the U. S. Government.

Management Sciences Research Group
Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

[illegible] has been approved for public release and sale; its distribution is unlimited.

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. SPECIAL
A

A SINGLE SOURCE TRANSPORTATION ALGORITHM

by

Robert V. Nagelhout and Gerald L. Thompson
Carnegie-Mellon University

ABSTRACT

A single source transportation problem is an ordinary transportation problem with the additional requirement that the entire demand at each demand location be supplied from a single supply location. It is a special case of Ross and Soland's generalized assignment problem. Such problems occur frequently in applications. This paper gives two heuristic solution methods and a branch and bound algorithm for solving single source transportation problems. A discussion of the branching rules, variable fixing rules, and the computation of weak lower bounds is given. Computational experience with the solution of randomly generated problems having up to 40,000 integer variables is reported.

1. INTRODUCTION

In this paper we consider ordinary transportation problems with the additional restriction that each demand must be entirely supplied from a single source. It is therefore a special kind of generalized assignment problem in the sense of Ross and Soland [5].

There are many applications in which such requirements are made on the solution. For instance, the supplying of supermarket orders from a network of central warehouses frequently has this restriction. In military applications it is common to require that all troops going on the same mission leave from the same staging area. When a group of computers is used

to fill a set of computation demands, the common requirement is that all computation on each single job be performed entirely by a single computer [1]. Many facility location models contain a single source requirement on the shipment of goods from the opened facilities to the demand locations [6]. Single source transportation problems are also related to a large class of "loading" or "packing" type problems where we attempt to assign a set of weighted objects to boxes or bins which have weight capacities [2].

In [3] De Maio and Roveda formulated a problem which is a special version of the generalized assignment problem stated later by Ross and Soland [5]. Srinivasan and Thompson [7] showed how to transform De Maio and Roveda's problem into a single source transportation problem. They proposed solving the latter using a branch and bound cost operator algorithm which used the ordinary transportation problem as a relaxation of the single source problem.

In the present paper we describe and give computational results for two heuristic solution methods and a cost operator algorithm which is similar to the algorithm described in [7]. The present algorithm differs from that in [7] in the following respects: (a) We have replaced the "row unique" solution concept in [7] by our "single source" solution concept, see section 5; (b) Different variable selection and branching rules are used. (c) Weak lower bounds are calculated and are used in fathoming as well as for variable selection. (d) A non-basic variable fixing rule has been added.

In section 6 computational results from the solution of problems ranging in size from 5 x 10 to 100 by 400 are presented. All of these problems were generated randomly using the method described in Ross and Soland [5]. A discussion is given concerning the way that problem difficulty depends on the setting of parameters in the Ross and Soland problem generator.

2. STATEMENT OF THE PROBLEM

In the single source transportation problem we consider a set of sources $I = \{1,\ldots,m\}$ each having capacities $a_i > 0$; a set of uses (or users) $J = \{1,\ldots,n\}$ each having known demands $b_j > 0$; and a set of costs c_{ij} of supplying use j from source i. The problem is to assign sources to uses so that: (i) the total amount shipped from each source does not exceed its capacity; (ii) each use is supplied by exactly one source; and (iii) the total cost Z of the assignment is minimized.

We shall assume, as a necessary but not sufficient condition, for feasibility that

$$\sum_{i\in I} a_i \geq \sum_{j\in J} b_j \tag{1}$$

i.e., that supply exceeds or equals demand. Defining $J' = J \cup \{n+1\}$, $c_{i,n+1} = 0$ for $i\in I$, $b_{n+1} = \sum_{i\in I} a_i - \sum_{j\in J} b_j$, and letting x_{ij} be the amount shipped from i to j, we can write the single source transportation problem as:

$$\text{Minimize } Z = \sum_{i\in I}\sum_{j\in J} c_{ij}\, x_{ij} \tag{2}$$

Subject to

$$\sum_{j\in J'} x_{ij} = a_i \quad \text{for } i\in I \tag{3}$$

$$\sum_{i\in I} x_{ij} = b_j \quad \text{for } j\in J' \tag{4}$$

$$x_{ij} \geq 0 \quad \text{for } i\in I,\ j\in J' \tag{5}$$

$$x_{ij} = 0 \text{ or } b_j \quad \text{for } j\in J \tag{6}$$

The problem defined by (2)-(5) is an ordinary transportation problem which we will call problem P. We denote by P' the problem in (2)-(6). We will say that problem P is the transportation relaxation of problem P'.

REMARK: In the case where all of the demands, b_j $j \in J$, are equal, P' can be transformed into an equivalent (semi-assignment) problem, P^*, having integral supplies and all demands equal to one. This can be done by making the following transformations: let $b_j = b$ for all $j \in J$

$$c^*_{ij} = c_{ij} \times b \qquad i \in I \quad j \in J' \qquad (7)$$

$$a^*_i = d_i \qquad i \in I \qquad (8)$$

where $$a_i = d_i \times b + k_i \qquad 0 \leq k_i < b \qquad (9)$$

$$b^*_j = 1 \qquad j \in J \qquad (10)$$

$$b^*_{n+1} = \sum_{i \in I} a^*_i - n \quad . \qquad (11)$$

The reader can verify that by making the transformations (7) - (11) the resulting single source transportation problem, P^*, is equivalent to the original problem, P', in the sense that X^* is a solution to P^* if and only if $X' = bX^*$ is a solution to P'. The significance of transforming P' into P^* is that P^* is a transportation problem with integral supplies and unit demands. Such problems are known as semi-assignment problems. Thus due to the well known unimodularity property of the basis matrix for a transportation problem, we know that the solutions to the transportation relaxation of P^* will either be 0 or 1 and therefore will satisfy the single source requirement automatically without any special search algorithm. In this paper we consider the case of unequal demands.

DEFINITION 1. By a single source basic solution to the transportation problem P we mean a feasible basic solution with the property that for each $j \in J$ there is a row index $i = i(j)$ such that $x_{ij} = b_j$; in other words a solution in which each demand is completely supplied by a single source.

In [7] the idea of row unique solutions were introduced, which are a special kind of single-source solutions. A row unique basic solution to problem P is a basic feasible solution with the property that for each $j \in J$ there exists a unique row $i' = i(j)$ such that $x_{i'j}$ is the only basic variable in column j. For a nondegenerate transportation problem, these two concepts are identical. In the case of (primal) degenerate problems, it is necessary to use the single-source solution concept instead of the row unique concept. We will elaborate on this point in Section 5. Since many of our problems are degenerate we concentrate on the former concept here.

Any single source solution to the transportation problem P gives rise to a feasible solution to problem P'. In Section 4 we describe a branch and bound algorithm which solves P' by finding single-source solutions to a series of transportation problem relaxations of P'. Each of these relaxed problems differs from P by having the cost of some of the cells set equal to $+M$ or $-M$ (where M stands for a number much larger than the absolute value of any cost) in order that those cells are forced into or out of the basic solution for the corresponding relaxed problem.

3. HEURISTIC SOLUTION METHODS

In order to reduce the size of the search tree in the branch and bound search process we developed two heuristic solution methods that almost

invariably find feasible solutions to P' in a short time. The smallest value of the objective function for such heuristic solutions is used as an initial upper bound in the branch and bound algorithm. In many cases these heuristic methods actually find an optimal solution, as will be discussed in Section 6 where data from problem solutions is given.

The two heuristic solution methods we used differ only in the order in which uses (columns) were selected to be assigned to sources (rows). The first method calculates "regrets" similar to those of the VAM starting solution method [4] for transportation problems; the second method selects uses in the order of non-increasing demand sizes.

To describe the regret heuristic let $c_{j_k j}$ be the kth smallest cost in column j. Define Reg(j), the regret for use j to be

$$\text{Reg}(j) = (c_{j_2 j} - c_{j_1 j})b_j \tag{12}$$

The first use to be assigned to a source is one whose regret is largest. Once it has been assigned (in a manner to be discussed in the next paragraph), it is removed from the set U of unassigned uses, new regrets are calculated as necessary, and a second use with largest regret is chosen to be assigned, etc.

Given that use j is to be assigned, we first construct a set S_j of sources to which it can be assigned, according to the following rule:

$$S_j = \{k \in I \mid c_{kj} - c_{j_1 j} \leq \sigma_j\} \tag{13}$$

where σ_j is a parameter chosen by some rule such as

$$\sigma_j = c_{j_3 j} - c_{j_1 j} \tag{14}$$

In other words S_j consists of the indices of all sources whose costs in column j differ from the smallest cost in column j by an amount less than or equal to σ_j. Then one of the indices $i \in S_j$ is chosen randomly to be the actual source to supply demand b_j.

The reasons for the random choice among indices in the set S_j are: first, with a specific choice rule it is quite easy to make early choices which lead to infeasible solutions; and second, with the random choice rule we can repeat the heuristic choice rule several times and retain the smallest cost solution found.

In order to state the regret heuristic more precisely we first define the notation to be used.

U = set of unassigned uses

Z = the cost of current solution (or partial solution)

S_j = set of all sources $k \in I$ such that $c_{kj} - c_{j_1 j} \leq \sigma_j$.

σ_j = a parameter whose size determines the number of elements in S_j. (In some problems it may be desirable to have σ_j change as more steps are taken in the algorithm.)

(H1) Regret heuristic.

(1) (Initialization) Let $U = J$, $Z = 0$. For each $j \in U$ find j_1, j_2, and j_3. Set $\sigma_j = c_{j_3 j} - c_{j_1 j}$.

(2) (Check feasibility) If $c_{j_1 j} = M$ for some $j \in U$ go to (8). Else go to (3).

(3) (Choose the use having the largest regret). Choose $j \in U$ such that $\text{Reg}(j) = (c_{j_2 j} - c_{j_1 j})b_j$ is a maximum.

(4) (Determine the choice set.) Calculate S_j which is defined in (12).

(5) (Select a source.) Choose $i \in S_j$ at random. Make the following replacements:

$$U \text{ by } U - \{j\}$$

$$Z \text{ by } Z + c_{ij} b_j$$

$$a_i \text{ by } a_i - b_j .$$

(6) If $U = \emptyset$ go to (9). Else go to (7).

(7) (Update the costs.) For all $h \in U$ if $b_h > a_i$ set $c_{ih} = M$. Find the two lowest cost cells $c_{h_1 h}$, $c_{h_2 h}$, in each column. Go to 2.

(8) Current solution is infeasible. Stop.

(9) Feasible solution found. Stop.

The second heuristic, which we call the "largest demand heuristic," is identical to the regret heuristic except that in step (3), instead of choosing at each step the use with the next largest regret, we choose the use with the next largest demand. Thus in the largest demand heuristic (H2), step 3 is replaced by:

(3') (Choose the next use.) Let $j \in U$ be the index of an unassigned use whose demand, b_j, is largest.

Since step (3) requires more effort in the regret heuristic than in the largest demand heuristic we find, as expected, that the regret heuristic uses more CPU time. However, neither of the heuristics requires very much time. For example, five runs of the regret heuristic on a 75 x 200 problem requires about 4 seconds of CPU time (DEC 20 computer).

The regret heuristic, being the "greedier" heuristic, tends to generate lower cost feasible solutions than the largest demand heuristic. The largest

demand heuristic was designed to find good feasible solutions in the case when the regret heuristic failed to find any solution. In all of our computational experience this phenomenon occurred only once; yet it may be more frequent in problems with a different data structure. We might add at this point that neither of the heuristics can be guaranteed to find a feasible solution to a given problem. It can be easily verified that the problem of finding a feasible solution to P' is NP complete. Finding a feasible solution to P' is equivalent to determining whether one can find a partition, $S_1,\ldots,S_m$, of a set of integers $\{b_1,\ldots,b_n\}$ (the demands) such that the sum of the elements in S_i equals a_i (the supply) for $i = 1,\ldots,m$. The latter problem is known to be NP complete. Thus it is probably necessary to carry out a partial enumeration of assignments such as is done in the method of section 4, in order to guarantee the finding of a feasible solution to P', when such a solution in fact exists.

4. THE COST OPERATOR ALGORITHM

We now describe a branch and bound algorithm which uses cost operators in the sense of Srinivasan and Thompson [8]. The basic idea in the algorithm is that the solution of problem P' is obtained by solving a sequence of related transportation problems $P, P_1,\ldots,P_t$ until we have found a problem, say P_k, such that the optimal transportation solution to P_k is also an optimal single source solution to P'. Each successive transportation problem differs from its predecessor in that certain costs have been changed from their original values to either $-M$ or $+M$. When we drive a cost to $-M$ we will say we have "fixed in" the corresponding cell, and when we drive it to $+M$ we will say that we have "fixed out" the cell. We now give more precise definitions.

We shall say that cell (i,j) has been fixed out of the basis when a cost operator has been applied to the problem so that c_{ij} becomes equal to +M, where M is so large that $x_{ij} = 0$ for all optimal solutions to the new problem. The operation of freeing a cell (i,j) which has been fixed out is the application of a cell operator to drive c_{ij} back to its original value.

We shall say that cell (i,j) has been fixed in the basis when

(a) a cost operator has been applied to the problem so that the cost c_{ij} becomes equal to -M, where M is so large that $x_{ij} = b_j$ in any optimal solution to the new problem;

(b) a_i has been replaced by $a_i - b_j$;

(c) for any k such that $b_k > a_i$ we fix cell (i,k) out of the basis, since source i cannot supply the demand at k.

By freeing a cell (i,j) which has been fixed in we mean to undo the actions listed above, so that the costs and rims go back to their previous values.

In (A) - (D) below we describe the various steps of the algorithm. The algorithm is stated in detail in (E) and a simple example is worked in (F).

(A) Tree Search Rules

The branch and bound algorithm which we are going to describe is of the LIFO (last in first out) or depth first variety. The method starts by first solving the problem P. If its solution satisfies the single source property we are finished. If not, we apply a nonbasic variable fixing rule, to be described later, which fixes some of the nonbasic variables at zero. Next a cell is selected by one of the cell selection rules given later, and the cell is fixed in the basis. This procedure is continued until either (a) a single source solution is obtained or (b) the objective function value of the new problem exceeds the current upper bound. In case (a) we compare the solution

with the current best single source solution and update the latter and the current upper bound if the current solution is better. In either case we backtrack by freeing the last cell fixed in. Then (using the LIFO rule) we choose the last cell fixed in and consider it for fixing out. If the objective function value of the current relaxation plus a weak lower bound (to be described later) exceeds the current upper bound, then we continue backtracking. Otherwise the cell is fixed out and the search process continues deeper in the search tree. A more precise description will be given later in this section.

(B) Nonbasic Variable Fixing Rule

At the initial node of the search tree the transportation problem P is solved. If the solution to P is not a single source solution then it is possible to examine the nonbasic variables in the solution to P in order to fix some of them at zero. Let Z_p, Z_u be the objective function value of P, and the current upper bound respectively. Let u_i and v_j be optimal dual variables associated with the current basic solution to P. For any nonbasic variable (i,j), $i \in I$, $j \in J$, if

$$(c_{ij} - (u_i + v_j)) * b_j \geq Z_u - Z_p - 1 \qquad (15)$$

then we can set $x_{ij} = 0$ for the remainder of the procedure. This is true since if a nonbasic variable x_{ij} were to become positive then it can only assume the value b_j, in which case the left hand side of (15) represents the minimum amount by which the objective function value of P will increase if we were to require that $x_{ij} = b_j$. Thus if this amount exceeds the current gap between the upper bound and the transportation relaxation then we can be sure that x_{ij} will not be positive in any single source solution to P'. Actually this variable fixing test could be repeated at any node of the search

tree, in which case Z_p would be replaced by the objective function value of the current relaxation. However, the computational cost of doing so would probably exceed the benefit.

It is interesting to note that the left hand side of (15) increases as the demand, b_j, increases. Intuitively this seems plausible since forcing a larger amount of demand into a single cell causes more of a restriction on the problem. In other words the constraint $x_{ij} \geq b_j$ becomes stronger as b_j increases. We have found that the variable fixing rule (15) will set to zero as many as 95% of the nonbasic variables. When solving problems in which 60% or more of the nonbasic variables are fixed to 0 by the variable fixing rule, it would probably be worthwhile to use a code which takes advantage of the sparse structure of the resulting transportation problem. We have not so far used such a sparse code in our studies.

(C) Weak Lower Bounds

The algorithm that we have implemented is essentially a linear programming based implicit enumeration scheme. At any node of the search tree we have solved a relaxation of problem P' where a variable, x_{ij}, is either fixed at zero, fixed at b_j, or free to assume any value between zero and b_j. If the objective function value of the current problem, Z_c, does not exceed the current upper bound, Z_u, and its solution is not single source, then we choose one of the free variables, say x_{ij}, and we create two new problems. In one of the new problems we require that $x_{ij} = 0$ and in the other problem we require that $x_{ij} = b_j$. Let us denote their objective function values by Z_0 and Z_b respectively. By calculating weak lower bounds we can determine the minimum amounts by which Z_c must increase if we require that $x_{ij} = 0$ or $x_{ij} = b_j$. Thus for any free variable x_{ij}, we can calculate a weak lower bound WLB_0 in the $x_{ij} = 0$ direction and WLB_b in the $x_{ij} = b_j$ direction where,

$$Z_c \leq Z_c + WLB_0 \leq Z_0 \tag{16}$$

or

$$Z_c \leq Z_c + WLB_b \leq Z_b \quad . \tag{17}$$

What we hope to find is that either

$$Z_c + WLB_0 \geq Z_u \tag{18}$$

or

$$Z_c + WLB_b \geq Z_u \tag{19}$$

or both. In case (18) ((19)) we can fathom the search tree in a particular direction without having to calculate Z_0 (Z_b). Of course there is a cost-benefit tradeoff involved here since some computational effort is required to calculate the weak lower bounds. Thus the real question is whether the extra information obtained from calculating the weak lower bounds is worth the added computational effort. We have found that in almost all of the tests which we have performed the weak lower bounds have proven to be cost effective. The weak lower bounds are useful not only for fathoming nodes in the search tree, but for choosing a "good" variable for branching. We will describe the branching rules which we have selected in the next section.

We now describe how the weak lower bounds are calculated. Suppose that at a given search tree node we wish to calculate a weak lower bound, WLB_0, in the direction $x_{ij} = 0$. This is done by applying a positive cell cost operator δc^{+}_{ij} to the cell (i,j) which determines the maximum amount, μ^{+}, by which the cost in cell (i,j) can be increased while maintaining optimality in the current basis. This is called the positive basis preserving cell cost operator, and

is described in [8]. At this point cell (i,j) contains some flow, x_{ij}, between zero and b_j and the weak lower bound WLB_0 is,

$$WLB_0 = \mu^+ x_{ij} \tag{20}$$

WLB_0 is a lower bound on amount by which the objective function value of the current relaxation will increase if we branch in the direction $x_{ij} = 0$. For a proof see [9].

In a similar but slightly more difficult manner we can calculate a weak lower bound, WLB_b, in the direction where $x_{ij} = b_j$. In this case we apply a negative cost cell operator δc_{ij}^- to cell (i,j) which determines the maximum amount, μ^-, by which the cost in cell (i,j) can be decreased before the current basis becomes non-optimal. If the cost in cell (i,j) is changed to c_{ij}^- where

$$c_{ij}^- = c_{ij} - \mu^- \tag{21}$$

then some nonbasic cell, say (p,q), will have a reduced cost, $c_{pq} - (u_p + v_q)$, of zero. If cell (p,q) is added to the basis then a unique "cycle" or "loop" is formed in the current basis tree. From this cycle we can determine a cell (e,f) which would leave the basis, and the amount Δ, by which the flow along this cycle would change if cell (p,q) were brought into and cell (e,f) were removed from the basis. The valid weak lower bound WLB_b is (see [9])

$$WLB_b = \mu^- \Delta \tag{22}$$

Notice that WLB_b is more difficult to obtain than WLB_0 since in the former case we must find the minimum giver amount, Δ, along the cycle created by an incoming nonbasic cell.

The effectiveness of the weak lower bounds depends for the most part on how close the current upper bound, Z_u, is to the value of the current relaxation, Z_c. If $Z_u - Z_c$ is small then (18) and (19) provide strong fathoming devices. Thus the ability of the heuristics to generate low cost feasible solutions is a crucial factor in the performance of the branch and bound algorithm. Another important factor is the branching or cell selection rules which we describe next.

(D) Cell Selection (Branching) Rules

We have tested several of many possible cell selection rules and have subsequently reduced our choices to two rules. These rules utilize both the regret concept and one of the weak lower bounds, WLB_0, discussed earlier. In order to describe these rules we first define the following quantities at any step of the tree search:

F = the set of columns containing more than one basic cell having positive flow

U = the set of columns containing no fixed in cells

B^+ = the set of basic cells having positive flow

WLB_{0i} = the weak lower bound in direction $x_{ij} = 0$ for cell (i,j).

Using this notation we describe two branching rules:

(1) BC-(basic cell) regret rule

a) Column selection - choose $j \in F$ such that $Reg(j) \geq Reg(k)$ for all $k \in F$, that is, choose a column having largest regret.

b) Row selection - find the two smallest cost cells (i_1,j), $(i_2,j) \in B^+$ and choose cell (i,j) where $i = i_1$ if $WLB_{0i_1} \geq WLB_{0i_2}$, and otherwise $i = i_2$; that is, of the two smallest cost cells in column j which have positive flows, pick the cell whose weak lower bound is largest.

2) G-(General) regret rule

a) Column selection - Choose $j \in U$ such that $Reg(j) \geq Reg(k)$ for all $k \in U$. (This rule is the same as (1) (a).)

b) Row selection - find the two smallest cost cells (i_1,j), $(i_2,j) \in I$ and choose (i,j) where $i = i_1$ if $WLB_{Oi_1} \geq WLB_{Oi_2}$, and $i = i_2$ otherwise; that is, of the two smallest cost cells in column j, pick the cell whose weak lower bound is largest.

It is clear that by using the G-regret rule there are more cells to choose than by using the BC-regret rule. Also the BC-regret rule only considers cells (i,j) whose flow x_{ij} satisfies $0 < x_{ij} < b_j$; we will call this a fractional flow, in agreement with the usual integer programming terminology. On the other hand, the G-regret rule may select a cell whose flow is b_j, i.e., an integral flow.

In most integer programming algorithms the cell selection criteria is limited to a choice among fractional flow variables. The reason is that if a non-constrained variable is "naturally" integer then it would seem to be unproductive (i.e., would increase the size of the search tree) to explicitly restrict the variable to be integer by branching on it. However, it is plausible that some of the variables which are naturally integer in the optimal solution to P might also have the same integer values in an optimal solution to P'. Hence it might be worthwhile to restrict these variables to their respective integer values in hopes that the size of the search tree will ultimately be reduced. In line with this reasoning we have found through computational experience (section 6) that in some cases the size of the search tree

is smaller when using the G-regret rule for branching rather than the BC-regret rule. The G-rule usually produces a smaller (in number of nodes) search tree than the BC-rule when the relaxation, P, has a solution which is not very close to being integer. That is when P contains only a few fractional flow cells, the BC-rule concentrates on the non-integral (i.e., the non single source) portion of the relaxation, and in doing so it is usually capable of performing a quick partial enumeration. However, when the problems are "tight," that is $\sum_{i \in I} a_i - \sum_{j \in J} b_j$ is small, and there are several fractional variables, it is often better to fix some of the variables (say those variables in columns having a large regret) into the basis even if they are not fractional. This avoids the possibility of these variables becoming fractional as we move deeper into the search tree. This is accomplished by using the G-regret rule. Clearly fixing non-fractional flow variables is not always the best alternative. However, we have noticed dramatic differences in the performance of the algorithm between the BC-rule and the G-rule on some of the more difficult problems which we have generated. For example on a particular 5 x 50 test problem (i.e., $m = 5$, $n = 50$) the search tree using the BC-rule had 13,814 nodes whereas the search tree using the G-rule had only 356 nodes. In our experience the BC-rule usually yields a search tree with fewer nodes than does the G-rule; however, the BC-rule is not invariably better than the G-rule.

(E) Statement of the Algorithm

Before we state the algorithm we define some notations:

ℓ = level of the tree

Z_ℓ = the transportation problem objective function value at level ℓ

X_ℓ = the transporation problem solution at level ℓ

U = the set of columns containing no cells which are fixed in

$T(\ell)$ = $(i,j)^-$ if cell (i,j) is fixed in at level ℓ, and = $(i,j)^+$ cell (i,j) is fixed out at level ℓ

(A1) The Single Source Cost Operator Algorithm

Step (1) (Heuristics) Run the regret and the largest demand heuristics to get an upper bound, Z_u. If neither gives a feasible solution then set $Z_u = \infty$.

Step (2) (Initialize) Let $U = J$; $\ell = 0$. Solve P. If $Z_0 \geq Z_u - 1$ go to (10). Otherwise go to (3).

Step (3) (Variable fixing) For all non-basic cells (i,j), if $(c_{ij} - (u_i + v_j)) * b_j \geq Z_u - Z_0 - 1$ then let $c_{ij} = M$ otherwise go to (4).

Step (4) (Cell Selection) Use either the BC-regret rule or the G-regret rule to choose a cell (i,j) upon which to branch. Save WLB_{0_i}.

Step (5) (Fix in cell) Replace ℓ by $\ell+1$. Let $T(\ell) = (i,j)^-$. Fix in cell (i,j). If $Z_\ell \geq (Z_u - 1)$ go to (6). Otherwise if X_ℓ is single source, save X_ℓ; let $Z_u = Z_\ell$; go to (6). Otherwise replace U by $U - \{j\}$ and go to (4).

Step (6) (Free after fixing in) Free cell (i,j); replace ℓ by $\ell-1$; go to (7).

Step (7) (Check WLB) If $Z_\ell + WLB_{0_i} \geq Z_u$ go to (9). Otherwise go to (8).

Step (8) (Fix out cell) Replace ℓ by $\ell+1$. Let $T(\ell) = (i,j)^+$. Fix out cell (i,j). If $Z_\ell \geq (Z_u - 1)$ go to (9). Otherwise if X_ℓ is single source, save X_ℓ; let $Z_u = Z_\ell$; go to (9). Otherwise go to (4).

Step (9) (Free after fixing out) Replace ℓ by $\ell-1$. If $\ell = 0$ then go to (10). Otherwise free cell (i,j). If $T(\ell) = (i,j)^+$ go to (9). Otherwise go to (8).

Step (10) Stop. The current solution, if there is one, is optimal. Otherwise there is no feasible solution to the problem.

(F) Example

Consider the problem given in Figure 1. Using the algorithm just described we will indicate the various steps involved in finding an optimal single source solution to this problem.

Step	Description of the Operation
1	Application of the regret heuristic with $\sigma_j = 1$ for all $j \in J$ yields $Z_u = 391$.
2	$U = \{1,2,3,4,5\}$; $\ell = 0$; The solution to P is given in figure 2(a) $Z_0 = 279.7$.
3	Apply the non-basic variable fixing rule and set $c_{13} = c_{24} = c_{43} = c_{44} = M$.
4	(using the BC regret rule) $F = \{2\}$ $i_1 = 1$; $i_2 = 2$. $WLB_{0_1} = 1.41 \times 16 = 22.6$; $WLB_{0_2} = 1.09 \times 2 = 2.18$. Choose cell (1,2).
5	Let $c_{12} = -M$; $c_{11} = M$; $c_{15} = M$; $\ell = 1$; $T(1) = (1,2)^-$. X_1 is a single source solution. See figure 2(b). Save X_1.
6	Let $c_{12} = .39$, $c_{11} = 2$, $c_{15} = 5.19$, $\ell = 0$.
7	$279.7 + 22.6 = 302.3 \not> 391$.
8	$\ell = 1$. $T(1) = (1,2)^+$. Let $c_{12} = M$. $Z_1 = 360.9$. X_1 is not a single source solution. See figure 2(c).
4	$F = \{2,5\}$. $Reg(2) = 111.6$, $Reg(5) = 37.9$. Choose column 2. $i_1 = 2$, $i_2 = 4$. $WLB_{0_2} = 21.46 \times 2 = 42.9$ $WLB_{0_4} = .04 \times 16 = .64$. Choose cell (2,2).
5	Let $c_{22} = -M$, $c_{21} = M$, $\ell = 2$, $T(2) = (2,2)^-$. X_2 is single source. See figure 2(d). $Z_2 = 458$.
6	Let $c_{22} = 10.2$, $c_{21} = 12.9$, $c_{24} = 30.1$; $\ell = 1$
7	$360.9 + 42.9 = 403.8 > 391$.
9	$\ell = 0$.
10	Stop. $Z = 391$ is optimal. $x_{21} = 12$, $x_{12} = 18$, $x_{23} = 7$, $x_{34} = 10$, $x_{45} = 26$.

Figure 2(e) contains the four node search tree for the example. The first number in the parenthesis at each node is the objective function value of the current relaxation and the second number is the current upper bound. At node one problem P is solved and the solution is not single source (see figure 2(a)). The upper bound at this point is 371 which was obtained by applying the regret heuristic with $\sigma_j = 1$ for all $j \in J$. At this point the non-basic variable fixing rule is applied to fix x_{13}, x_{24}, x_{43}, and x_{44} to zero. Cell (1,2) is chosen for branching by the BC rule and at node 2 a single source solution is obtained with a value equal to the upper bound. Next, using the LIFO rule, we move to node 3 where the solution again is not single source. Then we choose variable (2,2) and move to node 4. At node 4 the value of the relaxation exceeds the current upper bound thus we fathom the tree at node 4. Notice that the weak lower bound was used at node 3 so that it was not necessary to fix out cell (2,2).

5. SINGLE SOURCE VERSUS ROW UNIQUE SOLUTIONS

In [7] the concept of a row unique solution was used to characterize the acceptable solutions to the sources to uses problem. In the present paper we have replaced the idea of a row unique solution by a single source solution. Clearly every row unique solution is also a single source solution, but the converse is not true, as can be seen in the example shown in Figure 3(a). The basic solution shown there is a single source optimal solution (which is not a row unique solution) and therefore solves the single source transportation problem. However, the reader can verify that the primal solution shown in Figure 3(a) is unique, and that there is no basic optimal row unique solution to the problem. Hence, to find a row unique optimal solution, it is necessary to generate a branch and bound search tree such as the five node search tree shown in Figure 3(b).

This tree was generated by using the cost operator algorithm described in section 4(E). Because the search tree has only one node when we look for a single source solution, and five nodes when we look for a row unique solution, it is clear that use of the single source solution concept yields considerable computational savings even on this small problem and hence much larger savings would be expected on larger problems. Thus the single source concept is essentially a way of circumventing one of the problems created by primal degeneracy.

6. COMPUTATIONAL RESULTS

In this section we will discuss the computational performance of the heuristics (Sec. 3) and the cost operator algorithm (Sec. 4) on a set of randomly generated problems ranging in size from 5 x 20 to 100 x 400. These problems were generated in the manner described in [5] as follows. We use a uniform probability distribution to generate random integer costs c_{ij} for $i \epsilon I$, $j \epsilon J$, between 1 and 50; we let $c_{i,n+1} = 0$ for $i \epsilon I$; similarly we generate random integer demands, b_j for $j \epsilon J$, between 5 and 20. Then recalling that j_1 is the smallest entry in column j, we set

$$x_{j_1 j} = b_j \quad \text{and} \quad x_{ij} = 0 \quad \text{for} \quad i \neq j_1, \quad \text{all} \quad j \epsilon J \tag{23}$$

We then calculate the largest supply,

$$S = \max_{i \epsilon I} \{S_i | S_i = \sum_{j \epsilon J} x_{ij}\} \tag{24}$$

which is needed to guarantee feasibility of the solution in (23).

Finally for each $i \epsilon I$, let the supply $a_i = \alpha S$ where the slack parameter α is chosen to be less than 1; and set $b_{n+1} = \sum_{i \epsilon I} a_i - \sum_{j \epsilon J} b_j$.

As was mentioned previously, $b_{n+1} > 0$ is a necessary but not a sufficient condition for the existence of a feasible solution to P'. In general the smaller the value of α and hence of b_{n+1}, in a problem, the more difficult it is to solve. When the slack, b_{n+1}, is small, there is more incentive to divide the flow among the cells in a given column, and thus violate the single source constraints. Thus by varying the value of α we can make the slack for a given problem larger or smaller. For $\alpha = 1$, an optimal solution is given by the assignment in (23). For $\alpha < 1$, the assignment in (23) is infeasible and thus the single source problem is likely to be nontrivial.

Figure 4 contains the test results of 27 problems ranging in size from 5 x 20 to 75 x 200. All of these problems were solved with a code written in Fortran IV using the BC-regret rule discussed in section 4(D) on a DEC-20 time sharing system at CMU. The solution times in Figures 4 and 5 are subject to some measurement error due to variable loads on the time sharing system. A value of $\sigma_j = (c_{j2} - c_{j1})/2$ for all $j \epsilon J$ was used in the regret heuristic. For each problem the regret heuristic chose the best feasible solution out of the five trial solutions which it generated. We decided not to use the largest demand heuristic on these problems since the regret heuristic always found at least one feasible solution; however the largest demand heuristic may be useful in other problems having a different data structure.

Notice that each problem size is repeated three times. This is because the difficulty of these problems can vary greatly even for problems having the same dimension. Thus we felt it would be more informative to solve three test problems of a given size in order to demonstrate the potential variation in problem difficulty.

In addition to the problems shown in Figures 4 and 5 we solved three test problems given to us by Prof. Terry Ross. Our solution times, after taking into account the difference in computers, are comparable if not slightly better than those obtained by the Ross and Soland algorithm.

The regret heuristic error was evaluated on the basis of a percent error formula which is,

$$\text{percent error} = \frac{Z_H - Z_{OPT}}{Z_{OPT} - Z_R} \times 100 \tag{25}$$

where Z_{OPT} is the optimal value, Z_H is the value obtained by the regret heuristic, and

$$Z_R = \sum_{j \in J} b_j \min_{i \in I} \{c_{ij}\} \tag{26}$$

Z_R is the smallest possible objective function value for P'. The purpose in subtracting Z_R from the denominator in (25) was to avoid the problem of "scaling" which is characteristic of network problems having a transportation structure. That is, we could add or subtract some positive constant, δ, to each of the costs in any column of the transportation problem without affecting the set of optimal single source solutions. Although this scaling does not affect the set of optimal solutions, it does however affect the value of an optimal solution so that a standard percent error formula such as

$$\frac{|Z_H - Z_{OPT}|}{Z_{OPT}} \times 100 \tag{27}$$

could be made larger or smaller by scaling the data. Thus we avoid this scaling problem by subtracting Z_R from the denominator in (25). For example if δ is

added to the cost in each cell in column j, then each of the parameters in (25) increases by $b_j \cdot \delta$ so that there is no net effect on the percent error. Also, problems for which $Z_{OPT} = Z_R$ are usually uninteresting and thus we attempt to avoid this situation in practice by making α small enough so that Z_R is not an optimal value. A discussion of the scaling problem for general integer programming problems is given in Zemel [10].

Note that the CPU time to run the heuristic varied approximately linearly with the problem size as $m \times n$ increases. Note also that the percent error of the heuristic tends to decrease as $m \times n$ increases. This paradoxical result would be even more impressive if we had used the standard error formula (27) instead of (25). The reason that the heuristic solution method gets better as $m \times n$ becomes large is probably due to the fact that larger problems tend to have more alternate optimal solutions, and there is the possibility that some of them have the single source property. In any case it is useful to know that a heuristically generated feasible solution to a large problem has a fairly high probability of being optimal in those cases in which it is not possible to prove optimality in a reasonable length of time.

The variable fixing rule (15) seems to be quite effective in fixing non-basic variables to zero. In many of the problems as many as 95 percent of the non-basic bariables are fixed at the initial node of the search tree. As we mentioned in section 4(B), if the variable fixing rule consistently eliminates more than say 60% of the non-basic variables then it would probably be beneficial to use a sparse code to solve the transportation relaxations. The effectiveness of the variable fixing rule on a given problem is better the closer Z_p and Z_u are to each other, because then rule (15) permits fixing out more non-basic variables.

We tested the cost operator algorithm both with and without the weak lower bound calculations and found that the weak lower bounds are indeed cost effective. They help by reducing the size of the search tree and by reducing the overall CPU time. As can be seen from Figure 4, the weak lower bounds can be used to fathom several of the nodes in the search tree which would otherwise have required explicit enumeration. Also since we found that search trees tend to be smaller when the weak lower bounds are used for branch selection than when they are not, we conclude that they provide good branch selections.

As expected, the single source transportation problems exhibit a much higher variance in their execution times than do ordinary transportation problems. The difficulty of a given single source problem depends upon many factors such as: m, n, the gap between $Z_{p'}$ and Z_p, the amount of slack (i.e. b_{n+1}), the cost, supply, and demand distributions, the problem density, and the number of fractional variables in the initial transportation relaxation. Given two problems of the same size the one having the larger gap is usually more difficult to solve. Problems in which the demands are all equal are easy to solve since, as we mentioned in section 2, partial enumeration is not necessary in this case. If all of the supplies, S_i $i \in I$, are greater than or equal to S, as defined in (24), then it is easy to see that the solution given by (23) is optimal. When many of the costs are very large, or equivalently when the problem is sparse, the number of low cost solutions is reduced which then facilitates the implicit enumeration. We have found that one of the best indicators of the potential difficulty of soving a single source transportation problem, P', is the number of fractional variables in the transportation relaxation P. A "large" number of fractional variables in a basic solution to P essentially means that P is not a close relaxation to P' and thus we expect that such problems will be difficult to solve.

It is possible to determine the range within which the number of fractional variables in a basic solution to P must lie. Consider the distribution of the basic cells in any ordinary transportation problem. Given an $m \times n$ problem, we add one slack column, $n+1$, to the problem which has no single source restriction. The total number of basic cells is $m + (n+1) - 1 = m+n$. Each of the $n+1$ columns must contain at least one basic cell which leaves $m - 1$ basic cells to distribute among the $n+1$ columns. The worst case (in terms of the number of fractional variables in the basic solution to P) occurs when $m - 1$ of the first n columns contain two positive flow basic cells each, in which case there are $2(m-1)$ variables which violate the single source criterion. The most desirable case occurs when the basic solution to P is single source, that is when all of the $m - 1$ "extra" basic cells either appear in column $n+1$, or appear among the first n columns and have a flow of zero (they are primal degenerate). Thus the number of variables which violate the single source criterion lies in the interval $(0, 2(m-1))$. Notice that this interval is independent of n. Thus we can increase the number of column locations in a given single source problem without affecting the upper bound on the number of fractional variables. In practice we have found that when a column contains more than one basic cell, then it usually contains two, or in any event at most three basic cells. Given this observation one would expect that the number of fractional variables would be approximately $2(m-1)$. However the number of basic cells which appear in column $n+1$ plus the number of primal degenerate basic cells appearing in the first n columns greatly reduces the number of fractional variables in a basic solution to P. We have found that the number of fractional variables is usually much smaller than $2(m-1)$.

Figure 5 gives the computational results for seven problems ranging in size from 100 x 100 to 100 x 400. These problems were generated as described earlier except that once the problem is generated, we remove a cell (or give

it a large cost) with probability .6. Thus approximately 60 percent of the cells have been eliminated from consideration. The maximum number of fractional cells which can occur in any of these problems is 2 x 99 = 198. The number of fractional variables encountered ranged from 13 to 46 and, as expected, the problems having the larger number of fractional variables were more difficult to solve. This is also true in figure 4 where the only problem which could not be solved within 10 minutes of CPU times was a 75 x 200 problem with 69 fractional variables out of a possible 2 x 74 = 148. Thus we believe that number of fractional variables in the solution to P is a good measure of the potential difficulty of a given single source problem. Utilizing the characteristics of a class of problems to determine how difficult they may be to solve is important since as we mentioned earlier there can be a large variance in the difficulty of single source transportation problems of the same size. In figures 4 and 5 we have reported the characteristics of each problem which we feel provides an adequate measure of their complexity.

If we look at the total CPU times in figures 4 and 5 we note the encouraging fact that, except for the one 75 x 200 problem which required more than 600 seconds to complete, the overall problem difficulty does not seem to increase with increasing problem size. (Of course, the total time, which includes input and output times, does increase with problem size.) Thus it appears that the single source transportation model can be useful in the solution of actual problems, especially if a user is willing to settle for a heuristic solution whenever total CPU running time becomes excessive.

7. CONCLUSIONS

We have discussed the design and computational testing of a cost operator algorithm for solving single source transportation problems. We observed that

the computational difficulty of a randomly generated test problem depended on m, n, the P' to P gap, the amount of slack, the supplies, the demands, the distributions of costs, the density of the problem, and the number of fractional variables in the initial solution to P.

The regret heuristic seemed to perform very well on these test problems. The maximum percent error was 41, (as calculated in formula (25)), and more importantly the performance of the regret heuristic improved as the problem size increased. Using the value of the regret heuristic as an initial upper bound, the cost operator algorithm solved each of 32 of the 34 test problems in less than 210 seconds of CPU time on a DEC-20 computer.

Two surprising observations were made: First, the regret heuristic rule tends to make smaller errors as the problem size, $m \times n$, increases. Second, the overall solution time (with one or two exceptions) does not tend to increase with problem size.

These two observations lead us to conclude that the single source transportation model has the potential of becoming an easily applied operations research tool, even though it is used to solve an NP complete problem.

2	.39	42.8	3.9	5.19	0	28
12.9	10.2	16.4	30.1	M	0	25
M	M	14.4	1.2	M	0	10
4.25	4	55.4	30.7	3.73	0	26
12	18	7	10	26	16	

Figure 1

12 (2)	16 (.39)	42.8	3.9	5.19	0	28
12.9	2 (10.2)	7 (16.4)	30.1	M	16 (0)	25
M	M	0 (14.4)	10 (1.2)	M	0	10
0 (4.25)	4	55.4	30.7	26 (3.73)	0	26
12	18	7	10	26	16	

Figure 2(a)

M	18 (-M)	M	3.9	M	10 (0)	28
12 (12.9)	10.2	7 (16.4)	M	M	6 (0)	25
M	M	0 (14.4)	10 (1.2)	M	0	10
0 (4.25)	4	M	M	26 (3.73)	0	26
12	18	7	10	26	16	

Figure 2(b)

12 (2)	M	M	0 (3.9)	16 (5.19)	0	28
12.9	2 (10.2)	7 (16.4)	M	M	16 (0)	25
M	M	14.4	10 (1.2)	M	0	10
4.25	16 (4)	M	M	10 (3.73)	0	26
12	18	7	10	26	16	

Figure 2(c)

12 (2)	M	M	0 (3.9)	26 (5.19)	0	28
M	18 (-M)	7 (16.4)	M	M	8 (0)	25
M	M	14.4	10 (1.2)	M	0 (0)	10
M	4	M	M	M	8 (0)	26
12	18	7	10	26	16	

Figure 2(d)

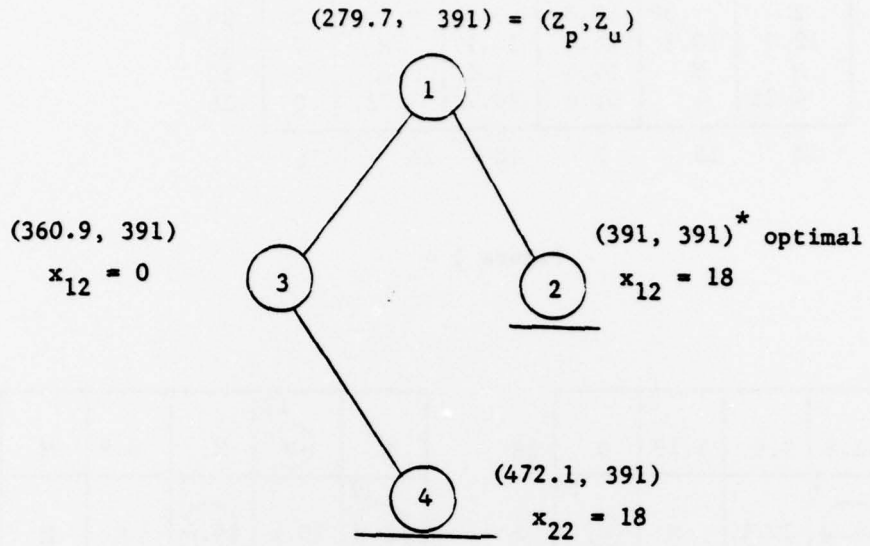


Figure 2(e)

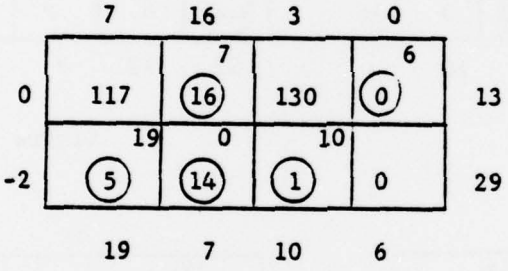


Figure 3(a)

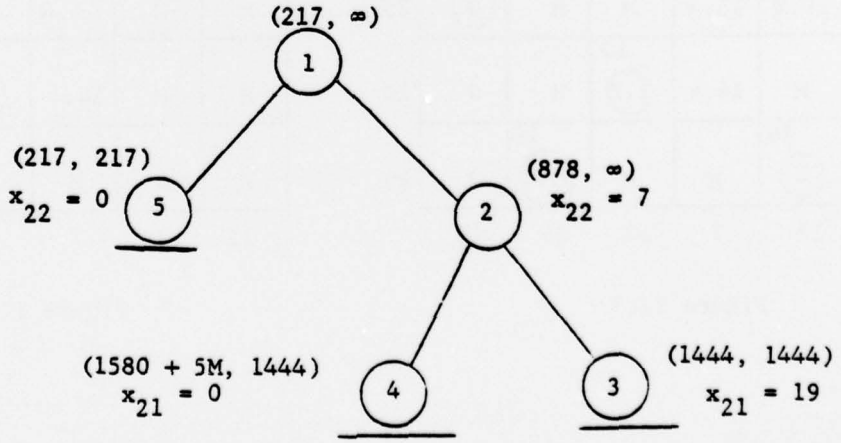


Figure 3(b)

m x n	Heuristics % error	Heuristics CPU* time	nodes	CPU* time Total	P to P' Gap	Z_{OPT}	Z_R	nodes fathomed by WLB	Vars. fixed by rule (15)	# of Fractional Variables	Value of α in Generator
5x20	24.5	.05	174	1.39	85	1881	1730	54	46	6	.7
5x20	0	.04	2	.16	33	2350	2287	2	74	2	.7
5x20	0	.04	3	.16	4	1913	1900	1	77	2	.7
5x100	36.9	.49	5611	137.1	28	10847	10533	2133	253	6	.7
5x100	9.9	.48	2461	68.6	28	11732	11421	597	340	6	.7
5x100	41.0	.49	19	1.3	5	10487	10409	9	365	4	.7
5x200	10.2	1.56	1766	83.4	14	18769	18573	700	728	4	.7
5x200	1.0	1.60	47	4.54	13	19861	19561	15	773	6	.7
5x200	20.7	1.60	2	2.56	0	21538	21258	0	692	4	.7
40x40	0	.41	7	.97	39	762	709	3	1425	8	.55
40x40	0	.41	23	.98	37	1009	913	7	1448	4	.55
40x40	0	.46	2013	42.0	81	943	818	403	1314	20	.55
40x125	11.5	1.54	177	10.16	9	2523	2497	87	4764	24	.55
40x125	0	1.52	2	3.06	6	2770	2757	2	4826	4	.55
40x125	0	1.49	173	9.73	5	2365	2360	73	4818	16	.55
40x200	14.2	2.65	27	6.44	2	4150	4115	19	7693	16	.55
40x200	0	2.65	76	8.86	9	3803	3784	32	7657	17	.55
40x200	3.5	2.71	3414	209.2	24	4358	4244	1392	7363	28	.55
75x75	5.1	1.46	861	27.0	29	1172	1133	291	5224	20	.50
75x75	7.5	1.42	804	28.5	38	1269	1203	240	5047	33	.50
75x75	0	1.12	0	1.19	0	1158	1158	0	0	16	.50
75x125	0	2.16	591	28.8	29	1989	1946	231	8738	49	.50
75x125	0	2.02	17	4.65	11	2137	2102	7	9051	48	.50
75x125	0	2.00	1	3.77	18	1953	1934	1	8916	38	.50
75x200**	-	4.34	7153	>600	--	--	3170	1848	13167	69	.45
75x200	0	4.12	2	8.5	7	2958	2942	1	14591	33	.45
75x200	0	4.03	15	9.24	17	3009	2979	9	14321	35	.45

All times in seconds on a DEC-20 computer.

* >Does not include Input/Output.

** Terminated after 10 minutes CPU time.

Figure 4

m X n	Heuristics % error	Heuristics CPU* time	nodes	CPU time Total	P to P' GAP	Z_{OPT}	Z_R	nodes fathomed by WLB	Vars. fixed by rule (15)	# of Fractional Variables	Value of α in Generator
100X100	0	2.11	505	23.88	90	2773	2640	159	9221	24	.6
100X150	0	3.30	946	48.07	60	3354	3251	362	14171	30	.50
100X200	3.3	4.38	772	49.63	75	4044	3867	290	18543	42	.50
100X250	14.08	5.73	78	16.02	9	5987	5916	32	24415	13	.55
100X300	0	7.71	20	18.75	14	6392	6356	8	29386	30	.55
100X350**	-	9.94	6749	>600	--	--	7036	2751	33486	46	.60
100X400	0	10.78	3	23.39	11	8236	8224	3	39292	20	.55

All times in seconds on a DEC-20 computer

* > Does not include Input/Output.

** Terminated after 600 sec.

Figure 5

REFERENCES

[1] Balachandran, V., "An Integer Generalized Transportation Model for Optimal Job Assignment in Computer Networks," Operations Research, Vol. 24, No. 4, pp. 742-759, July-August 1976.

[2] Christofides, N. and S. Eilon, "The Loading Problem," Management Sciences, 17, 259-268, 1971.

[3] Demaio, A. and C. Roveda, "An All Zero-One Algorithm for a Class of Transportation Problems," Operations Research, Vol. 19 (1971), pp.1406-1418.

[4] Gaver, D. P., and G. L. Thompson, Programming and Probability Models in Operations Research, Brooks/Cole, Monterey, Calif., 1973.

[5] Ross, G. T. and R. M. Soland, "A Branch and Bound Algorithm for the Generalized Assignment Problem," Mathematical Programming, Vol. 8 (1975) pp. 91-105.

[6] Ross, G. T. and R. M. Soland, "Modeling Facility Location Problems as Generalized Assignment Problems," Management Science, Vol. 24, No. 3, Nov. (1977) pp. 345-357.

[7] Srinivasan, V. and G. L. Thompson, "An Algorithm for Assigning Uses to Sources in a Special Class of Transportation Problems," Operations Research, Vol. 21 (1973), pp. 284-295.

[8] Srinivasan, V. and G. L. Thompson, "An Operator Theory of Parametric Programming for the Transportation Problem - I, II," Naval Research Logistics Quarterly, 19 (1972), pp. 205-252.

[9] Srinivasan, V. and G. L. Thompson, "Solving Scheduling Problems by Applying Cost Operators to Assignment Models," in Symposium on the Theory of Scheduling (S. Elmaghraby, Ed.) Berlin, N.Y., Springer-Verlag, (1973) pp. 399-425.

[10] Zemel, Eitan, "Functions for Measuring the Quality of Approximate Solutions to Zero-One Programming Problems," Discussion Paper No. 323, Graduate School of Management, Northwestern University, Evanston, Illinois 60201, June 1978.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(14) REPORT NUMBER: MSRR-429, WP-41-78-79

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

(6) 4. TITLE (and Subtitle): A SINGLE SOURCE TRANSPORTATION ALGORITHM

5. TYPE OF REPORT & PERIOD COVERED: Technical Report, February 1979

6. PERFORMING ORG. REPORT NUMBER: MSRR 429

(10) 7. AUTHOR(s): Robert V. Nagelhout, Gerald L. Thompson

(15) 8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0621

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR 047-048

11. CONTROLLING OFFICE NAME AND ADDRESS: Personnel and Training Research Programs, Office of Naval Research (Code 434), Arlington, Virginia 22217

12. REPORT DATE: Feb[illegible]979

13. NUMBER OF PAGES: 32

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

(12) 36p.

15. SECURITY CLASS. (of this report)

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

(9) Management sciences research rept.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Transportation Problem, Single Source Transportation Algorithm, Integer Programming Software, Cost Operator Algorithm, Generalized Assignment Problem.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A single source transportation problem is an ordinary transportation problem with the additional requirement that the entire demand at each demand location be supplied from a single supply location. It is a special case of Ross and Soland's generalized assignment problem. Such problems occur frequently in applications. This paper gives two heuristic solution methods and a branch and bound algorithm for solving single source transportation problems. A discussion of the branching rules, variable fixing rules, and the computation of weak lower bounds is given. Computational experience with the solution of randomly generated problems having up to 40,000 integer variables is reported.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

403 426 not